语义分析实验报告

1. 实验内容：

编写语义分析程序，实现对算术表达式的类型检查和求值。要求所分析算术表达式由如下的文法产生。



要求：

用自底向上的语法制导翻译技术实现对表达式的分析和翻译。

1. 写出满足要求的语法制导定义或翻译方案。
2. 编写分析程序，实现对表达式的类型进行检查和求值，并输出：
3. 分析过程中所有产生式。
4. 识别出的表达式的类型。
5. 识别出的表达式的值。
6. 实验方法：可以选用以下两种方法之一。
7. 自己编写分析程序。
8. 利用YACC自动生成工具。
9. 实验目的

了解体会自底向上的语法制导翻译技术，熟练使用lex和yacc两种工具

1. 实验过程

lex过程：

正则表达式获取数据

digit [0-9]

number {digit}+

decimal {digit}+(\.{digit}+)

将数据返回到yacc的几个token中。

{number} { int temp;

sscanf(yytext,"%d",&temp);

yylval = temp;

return NUM; }

{decimal} {double temp;

sscanf(yytext,"%lf",&temp);

yylval = temp;

return DECIMAL;

"\n" { return \* yytext; }

. { return \* yytext; }

yacc过程：

#define YYSTYPE double在yacc中YYSTYPE的缺省值是int,而在本次实验中我们可以选取double的数值，所以更改YYSTYPE

下面为针对每一条表达式所做的行为扩充每一个大括号里面保存的是C语言，作用是显示和控制

:expr'\n' {printf("运算结果为：%lf\n",$1);return 0;}

;

expr:expr'+'term {$$=$1+$3;printf("E -> E + T\t");printf("E = %lf + %lf = %lf\n",$1,$3,$$);}

|expr'-'term {$$=$1-$3;printf("E -> E - T\t");printf("E = %lf - %lf = %lf\n",$1,$3,$$);}

|term {$$=$1;printf("E -> T\t");printf("E = %lf\n",$1);}

;

term:term'\*'factor {$$=$1\*$3;printf("T -> T \* F\t");printf("T = %lf \* %lf = %lf\n",$1,$3,$$);}

|term'/'factor {$$=$1/$3;if($3==0) {yyerror("除零"); return 0;} printf("T -> T / F\t");printf("T = %lf / %lf = %lf\n",$1,$3,$$);}

|factor {$$=$1;printf("T -> F\t");printf("T = %lf\n",$1);}

;

factor:DECIMAL {$$=$1;printf("F -> NUM.NUM\t");printf("F = %lf\n",$1);}

|'('expr')' {$$=$2;printf("F -> (E)\t");printf("F = %lf\n",$1);}

|NUM {$$=$1;int temp;temp=$1;printf("F -> NUM\t");printf("F = %d\n",temp);};

1. 源代码

LEX:

%{

#include <string.h>

#include <stdio.h>

%}

digit [0-9]

number {digit}+

decimal {digit}+(\.{digit}+)

%%

{number} { int temp;

sscanf(yytext,"%d",&temp);

yylval = temp;

return NUM; }

{decimal} {double temp;

sscanf(yytext,"%lf",&temp);

yylval = temp;

return DECIMAL;}

"\n" { return \* yytext; }

. { return \* yytext; }

%%

int yywrap()

{

return 1;

}

YACC

%token NUM

%token DECIMAL

%{

#include <stdio.h>

#include <ctype.h>

#define YYSTYPE double//更改缺省的int

%}

%%

s:expr'\n' {printf("运算结果为：%lf\n",$1);return 0;}

;

expr:expr'+'term {$$=$1+$3;printf("E -> E + T\t");printf("E = %lf + %lf = %lf\n",$1,$3,$$);}

|expr'-'term {$$=$1-$3;printf("E -> E - T\t");printf("E = %lf - %lf = %lf\n",$1,$3,$$);}

|term {$$=$1;printf("E -> T\t");printf("E = %lf\n",$1);}

;

term:term'\*'factor {$$=$1\*$3;printf("T -> T \* F\t");printf("T = %lf \* %lf = %lf\n",$1,$3,$$);}

|term'/'factor {$$=$1/$3;if($3==0) {yyerror("除零"); return 0;} printf("T -> T / F\t");printf("T = %lf / %lf = %lf\n",$1,$3,$$);}

|factor {$$=$1;printf("T -> F\t");printf("T = %lf\n",$1);}

;

factor:DECIMAL {$$=$1;printf("F -> NUM.NUM\t");printf("F = %lf\n",$1);}

|'('expr')' {$$=$2;printf("F -> (E)\t");printf("F = %lf\n",$1);}

|NUM {$$=$1;int temp;temp=$1;printf("F -> NUM\t");printf("F = %d\n",temp);}

;

%%

#include "lex.yy.c"

int main(){

yyparse();

return 0;

}

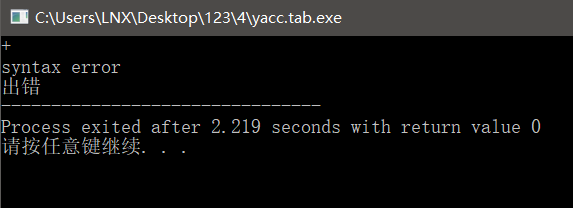
void yyerror(char \* s){

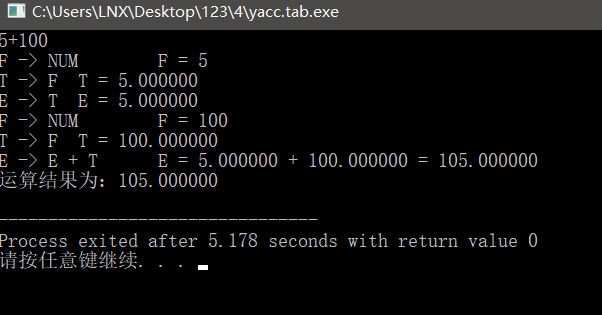
printf("%s\n",s);

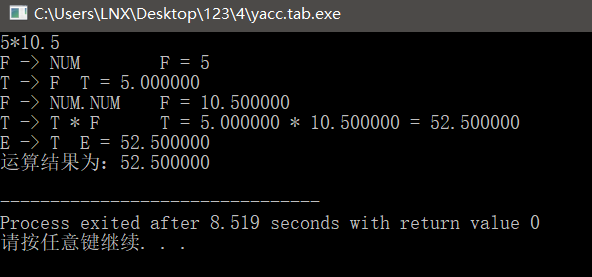
printf("出错");

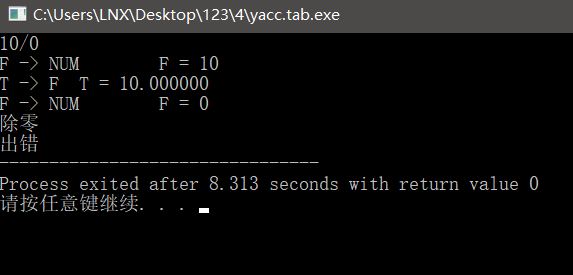
}

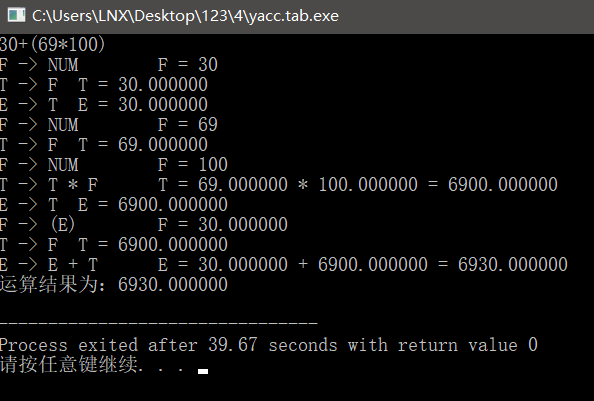
1. 测试样例











1. 总结分析

通过本次实验，加深了自底向上的语法制导翻译技术。